POTENTIAL GAME BASED COOPERATIVE CONTROL

IN DYNAMIC ENVIRONMENTS

A Thesis
Presented to
The Academic Faculty

by

Yusun Lee Lim

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
February 2011

POTENTIAL GAME BASED COOPERATIVE CONTROL

IN DYNAMIC ENVIRONMENTS

Approved by:

Dr. Jeff Shamma, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Ayanna Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: March 21011

I would like to dedicate this thesis to my parents for all their support and guidance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objectives of this research are to extend cooperative control methods based on potential games to dynamic environments and to develop an experimental test bed to illustrate theoretical results. Cooperative control concerns coordinating a collective performance of multiple autonomous agents. Possible applications include mobile sensor networks, distributed computation, and unmanned vehicle teams. Prior work has explored game theory, specifically the framework of potential games, as an approach to cooperative control, but has been restricted to static environments.

This research shows that potential game based cooperative control also can be applied to dynamic environment problems. The approach is illustrated on three example problems. The first one is a moving target tracking problem using a modified form of the learning algorithm "restrictive log-linear learning." The second example is mobile sensor coverage for an unknown dynamic environment. The last example is multi-agent path optimization using payoff based learning. The performances of the developed systems are studied by simulation. The last part of this thesis develops an experimental moving target tracking system using multiple mobile robots. Finally, the thesis concludes with suggestions for future research directions.

CHAPTER 1

INTRODUCTION

1.1 Cooperative control

The objective of cooperative control is to enable a group of interconnected systems to achieve a global or collective goal through local or individual controllers. The individual controllers make decisions based on local information and resources to achieve a collective objective of the whole system. This distributed decision architecture gives cooperative control the capacity for self-organization and robustness to uncertainties such as individual component failures and dynamic environments. On the other hand, cooperative control presents challenges such as the complexity associated with a large number of agents and the analytical difficulties of distributed local information. Nonetheless, prior work on cooperative control has explored a variety of application including mobile ad hoc networks [1], dynamic sensor coverage problem [2], and autonomous operation of a group of unmanned aircrafts or vehicles [3, 4]. Furthermore cooperative control research is also found under related titles such as multi-agent control, distributed systems, networked control, and swarming [5, 6, 7].

The most significant difference between cooperative control and centralize control is distribution of information. None of the agents in a cooperative control scenario can access all information in the system, and it is generally assumed that there is a communication cost for distributing local information. Therefore efficient and minimal communication is desired for a cooperative system.

1.2 Game theoretic approach

Game theory is a branch of economics that has its roots in the work of von Neumann and Morgenstern [8]. While the early motivation focused on two player zero-sum games, modern game theory broadens its scope to the study of interaction between self-interested agents [9].

In multi-agent systems both individual outcomes as well as the global outcome will depend on the decisions made by all agents in the system. This implies that in order for an agent to make the choice that optimizes its outcome, it must take into account the decisions that other agents may make while acting to optimize their own outcome. Game theory gives a method of examining such concerns [9].

A challenge in exploiting game theoretic concepts in multi-agent systems is that game theory traditionally was developed for the purpose *analyzing* social phenomena, whereas cooperative control applications concern *designing* engineered systems [5]. Analyzing existing societal phenomena calls for a descriptive way of thinking, but designing original engineered systems calls for a prescriptive view. Therefore it can be unclear how to implement conventional game theory to multi-agent systems directly.

1.3 Cooperative control based on Potential games

Recent research has explored the role of a special class of games for cooperative control, namely potential games [10]. For these games, a special alignment condition of individual utility functions and the global objective function guarantees the existence of at least one Nash equilibrium point in which the global objective function is maximized. Furthermore, one can design both utility functions that result in a potential game structure and learning algorithms that steer agents to a Nash equilibrium of the induced game. Learning algorithms can take into account the distributed topology of agents and limited communications.

One shortcoming of this approach is that it assumes a static environment and formulates a fixed game for this environment. Players' utilities and objective functions in a static environment are independent from time. However players' utilities in a dynamic environment is a function of time so that a Nash equilibrium can change over time. In that case, the conventional approach may not be enough to adapt players' action to the situation.

Motivated by this issue, this thesis explores modifying the existing potential game based multi-agent system design to address dynamic environments example. Furthermore, the new methods are tested through simulation and experimental implementation.

1.4 Overview of thesis

Chapter 2 presents some background on game theoretic concepts and potential game based cooperative control in static environments. Chapter 3 presents *a moving target tracking problem*, and shows that predicting future state can improve the system even though the prediction is not always correct. Chapter 4 presents *cooperative sensor coverage in unknown dynamic environments*, and illustrates how to simultaneously estimate and cover the unknown dynamic environment. Chapter 5 presents *payoff based path optimization problem* in stochastic environments. It demonstrates cooperative control of path rather than positions in dynamic environments without any communication between players. Chapter 6 presents an experiment of moving target tracking with mobile robot system and discusses some issues that are overlooked in the theory and simulations. Lastly, Chapter 7 summarizes the research and suggests further research topics.

CHAPTER 2

BACKGROUND

In this chapter, we will present a brief background of selected game theoretic concepts. See to [10] and [11] for a more review in depth.

2.1 Potential game

There is a set of players $\mathrm{P} = \{P_1, \cdots, P_n\}$. Each player $P_i \in P$ is assigned an objective function $U_i: A \to \mathbb{R}$ and an action set A_i where $A = \prod_{P_i \in P} A_i$. Let $a_i \in A_i$ denote an action of player P_i and a_{-i} denote a collection of actions of players other than P_i. Hence a overall joint action profile a is equivalent to (a_i, a_{-i}).

The definition of potential game [12] is as follows.

DEF. Potential game

Player action sets $\{A_i\}_{i=1}^n$ *together with player objective functions* $\{U_i: A \to \mathbb{R}\}_{i=1}^n$ *constitute a potential game if, for some function* $\phi: A \to \mathbb{R}$*,*

$$U_i(a_i'', a_{-i}) - U_i(a_i', a_{-i}) = \phi(a_i'', a_{-i}) - \phi(a_i', a_{-i}) \tag{2.1}$$

for every player $P_i \in P$*, for every* $a_i', a_i'' \in A_i$*, and for every* $a_{-i} \in$*. The function* ϕ *is called a potential function.*

The most significant characteristic of potential game is perfect alignment between the potential function and the players' local objective functions under unilateral changes.

In other words, if a player in a potential game changes its action unilaterally, the change of the player objective function and the potential function are equal.

DEF. Nash equilibrium

An action profile $a^* \in A$ is called a pure Nash equilibrium if for all players $P_i \in P$,

$$U_i(a_i^*, a_{-i}^*) = \max_{a_i \in A_i} U_i(a_i, a_{-i}^*) \tag{2.2}$$

Nash equilibrium is an important solution concept in game theory. If an action profile is a Nash equilibrium, there is no unilateral change of a player's action that can increase its local objective function. There may exist more than one Nash equilibrium in a game. It is easily proved that any action profile maximizing the potential function is a pure Nash equilibrium. Hence a potential game has at least one Nash equilibrium.

2.2 Illustration: Consensus as a potential game

In this section, the potential game concept is illustrated on a cooperative control problem. Consider a consensus problem with n-player set P where each player $P_i \in P$ has a finite action set A_i.

The global objective function for the consensus problem can be designed as

$$\phi(a) \coloneqq -\sum_{P_i \in P} \sum_{P_j \in N_i} \frac{\|a_i - a_j\|}{2} \tag{2.3}$$

where $N_i \subset P$ is player P_i's neighbor set. The potential function value is the sum of differences between all players. If an action profile reaches consensus then the potential function value will be 0. The next step is to assign each player an objective function that is completely aligned with the global objective. For the consensus problem with an

undirected and time-invariant interaction topology, player P_i can be assigned the utility function

$$U_i(a_i, a_{-i}) = -\sum_{P_j \in N_i} \|a_i - a_j\|. \tag{2.4}$$

This form of utility function is called wonderful life utility (WLU) [13]. Loosely speaking, we can view the WLU as equivalent to the change in global objective function that would have arisen if $P_i \in P$ "had never existed". Note that players' utility function (2.4) is only dependent on the actions of its neighbors. Now the global target function and individual objective function constitute a potential game.

Claim: Players' utility function (4) and global objective function (3) constitute a potential game provided that the time-invariant interaction topology induced by the neighbor sets $\{N_i\}_{i=1}^n$ *is undirected.*

Proof: The potential function can be expressed as

$$\phi(a) = -\sum_{P_j \in N_i} \|a_i - a_j\| - \sum_{P_j \neq P_i} \sum_{P_k \in N_j \setminus P_i} \frac{\|a_j - a_k\|}{2}. \tag{2.5}$$

The change in the objective function of player P_i by switching from action a_i^1 to action a_i^2 provided that all other players collectively play a_{-i} is

$$\begin{aligned} U_i(a_i^2, a_{-i}) - U_i(a_i^1, a_{-i}) &= \sum_{P_j \in N_i} \{\|a_i^1 - a_j\| - \|a_i^2 - a_j\|\} \\ &= \phi(a_i^2, a_{-i}) - \phi(a_i^1, a_{-i}). \end{aligned} \tag{2.6}$$

■

Therefore we can guarantee that there exists an action profile that is Nash equilibrium of the potential game, and the action profile is a solution of this consensus problem [12].

2.3 Learning algorithm

2.3.1 Binary restrictive log-linear learning

Log-linear learning for potential games was introduced by [14]. In log-linear learning, one player is randomly chosen with uniform probability over the player set at every time *t*. Only the selected player P_i is allowed to update its action and other players have to repeat their current actions. The chosen player P_i randomly updates its action according to the randomized strategy $p_i(t) \in \Delta(A_i)$. The a_ith component $p_i^{a_i}(t)$ of its strategy is

$$p_i^{a_i}(t) = \frac{exp\{\beta U_i(a_i, a_{-i}(t-1))\}}{\sum_{\bar{a}_i \in A_i} exp\{\beta U_i(\bar{a}_i, a_{-i}(t-1))\}} \tag{2.7}$$

for some arbitrary parameter $\beta \geq 0$. The parameter β determines player P_i's willingness to optimize its action. If $\beta = 0$, player P_i will update its action $a_i \in A_i$ with uniform probability over A_i. If $\beta \to \infty$, player P_i among its best response set with high probability. In potential games under log-linear learning the stationary distribution $\mu \in \Delta(A)$ of the joint action profile is

$$\mu(a) = \frac{exp\{\beta \emptyset(a)\}}{\sum_{\bar{a} \in A} exp\{\beta \emptyset(\bar{a})\}} \tag{2.8}$$

The distribution $\mu(a)$ can be interpreted as a probability that $a(t) = a$ for sufficiently large time *t*.

However a player's action set at time *t* might be restricted by its previous action at time *t-1*. To address this issue, binary restricted log-linear learning[1] was introduced by [10]. In binary restricted log-linear learning, available action set of P_i at time *t* is denoted as $R_i(a_i(t-1)) \subset A_i$. Then an action for next time step should be chosen among

1. Binary restrictive log-linear learning was introduced in [3] as 'binary Restrictive Spatial Adaptive Play (RSAP)'

$R_i(a_i(t-1))$ rather than A_i. Binary restricted log-linear learning can be described as follows: one player is randomly chosen with uniform probability over given player set at every time *t*. Only the selected player P_i is allowed to update its action and other players have to repeat their current actions as in log-linear learning. The selected player P_i randomly chooses one trial action $\hat{a}_i \in R_i(a_i(t-1))$ as follows:

$$Pr[\hat{a}_i = a_i] = \frac{1}{z_i} \quad \textit{for any action } a_i \in R_i\big(a_i(t-1)\big)\backslash a_i(t-1) \tag{2.9}$$

$$Pr[\hat{a}_i = a_i(t-1)] = 1 - \frac{|R_i(a_i(t-1))|-1}{z_i} \tag{2.10}$$

where $z_i = max_{a_i \in A_i}|R_i(a_i)|$. Then P_i decides its action at time *t* with the following probability:

$$Pr[a_i(t) = \hat{a}_i] = \frac{exp\,\{\beta U_i(\hat{a}_i, a_{-i}(t-1))\}}{D} \tag{2.11}$$

$$Pr[a_i(t) = a_i(t-1)] = \frac{exp\,\{\beta U_i(a(t-1))\}}{D} \tag{2.12}$$

where $D = exp\{\beta U_i(\hat{a}_i, a_{-i}(t-1))\} + exp\{\beta U_i(a(t-1))\}$ and $\beta \geq 0$ is an exploration parameter.

There are two assumptions regarding the restricted action set $R_i(a_i)$ for player $P_i \in P$. The first assumption is *reversibility*: for any player $P_i \in P$ and any action pair $a_i^1, a_i^2 \in A_i$, we impose $a_i^2 \in R_i(a_i^1) \Leftrightarrow a_i^1 \in R_i(a_i^2)$. The second assumption is *feasibility*: for any player $P_i \in P$ and any action pair $a_i^0, a_i^m \in A_i$, there exists a sequence of actions $a_i^0 \to a_i^1 \to \cdots \to a_i^m$ that satisfies $a_i^k \in R_i(a_i^{k-1})$ for all $k \in \{1,2,\cdots,m\}$. It is proved in [10] that if all players in a potential game implement restricted log-linear learning in a consensus problem with undirected and time-invariant interaction graph, and restricted action sets satisfying reversibility and feasibility, then the action profile *a* will maximize the potential function with arbitrary high probability at sufficiently large time *t*.

2.3.2 Sample Experimentation dynamics

The distinctive assumption of "payoff based" algorithms is that players can access their own realized payoffs only. In many practical systems, performance can be measured only after an action is implemented. Moreover players may not aware of exact payoff structure of the system. Therefore it is impossible to predict accurately the reward that would have been received from taking other actions.

An example of this assumption is distributed routing for ad hoc networks while minimizing the delay of packets to their destinations [15]. The most challenging difficulty is that each node has to rout packets to neighbors without knowledge of the entire network structure. Trial and error based tactics using observed reward history is reasonable for this type of problem.

Reference [11] introduced payoff based learning algorithm for potential games. These dynamics assure that agents' strategies asymptotically form a Nash equilibrium with arbitrarily high probability. The dynamics considered in this thesis is "Sample Experimentation Dynamics" from [11]. In Sample Experimentation Dynamics, a system converges to a Nash equilibrium with arbitrary high probability in any potential game even in the presence of utility measurement noise.

The algorithm of Sample Experimentation Dynamics is as follows. First, each player plays a random action, $a_i(0) \in A_i$ at time $t = 0$ and this action is set as the player's initial *baseline action*, i.e. $a_i^b(1) = a_i(0)$. Then each player starts an *exploration phase* over the next *m* periods. For convenience, double index the time when the actions are played as

$$a(t_1, t_2) = a(mt_1 + t_2) \tag{2.13}$$

where t_1 is the index of the exploration phase and t_2 indexes the played actions in the exploration phase. Let t_1 be the *exploration phase time* and t_2 be the *exploration action*

time. For any exploration phase time $t_1 \geq 1$ and any exploration action time $t_2 \in [1, m]$, each player selects its next action as follows;

- $a_i(t_1, t_2) = a_i^b(t_1)$ with probability $(1 - \varepsilon)$
- $a_i(t_1, t_2)$ is chosen randomly with uniform distribution over $(A_i \backslash a_i^b(t_1))$ with probability ε

After finishing the exploration phase, each player calculates average utility for every action taken during the phase. Let $n_i^{a_i}(t_1)$ be the number of times that player P_i played action a_i during the exploration phase at time t_1. Then the average utility for action a_i through the exploration phase at time t_1 is

$$V_i^{a_i}(t_i) = \begin{cases} \frac{1}{n_i^{a_i}(t_1)} \sum_{t_2=1}^{m} I\{a_i = \breve{a}_i(t_1, t_2)\} U_i(\breve{a}_i(t_1, t_2)), & n_i^{a_i}(t_1) > 0 \\ U_{min}, & n_i^{a_i}(t_1) = 0 \end{cases} \tag{2.14}$$

where $I\{\cdot\}$ is the usual indicator function and U_{min} satisfies

$$U_{min} = \min_i \min_{a \in A} U_i(a).$$

Define $A_i^*(t_1)$ to be the set of actions whose average utilities surpass the baseline action, i.e.

$$A_i^*(t_1) = \left\{ a_i \in A_i : V_i^{a_i}(t_i) \geq V_i^{a_i^b(t_1)}(t_i) + \delta \right\} \tag{2.15}$$

Each player will update its baseline action as follows.

- If $A_i^*(t_1)$ is empty, then $a_i^b(t_1 + 1) = a_i^b(t_1)$.
- If $A_i^*(t_1)$ is not empty, then
    - $a_i^b(t_1 + 1) = a_i^b(t_1)$ with probability ω
    - $a_i^b(t_1 + 1)$ is chosen randomly with uniform distribution over $A_i^*(t_1)$ with probability $(1 - \omega)$

Lastly, each player starts new exploration phase with updated baseline action and repeats these ‘exploration phase’ and ‘baseline update’ processes.

The following theorem is proved by [11].

Theorem

Let G be a finite n-player potential game where players' utilities are corrupted with a zero mean noise process. Assume all players use the Sample Experimentation dynamics. For any

- *probability* $p < 1$
- *tolerance level* $\delta \in (0, \alpha)$
- *inertia* $\omega \in (0,1)$
- *exploration rate* ε *satisfying* $min\left\{\frac{\alpha-\delta}{4}, \frac{\delta}{4}, 1-p\right\} > (1-(1-\varepsilon)^n) > 0$

and if the exploration phase length m is sufficiently large, then for all sufficiently large times $t > 0$, $a(t)$ *is a Nash equilibrium of G with at least probability p.*

Sample Experimentation Dynamics does not require any explicit cooperation or communication between agents. This is a desirable feature of these since it gives robustness to uncertainties in adaptive processes.

CHAPTER 3

TARGET TRACKING PROBLEM

3.1 Potential game in Dynamic environments

The cooperative control problems described in [10] and [11] assumes static environments. This thesis investigates cooperative control based on potential games for a dynamic environment. The main idea is estimating the next state of the environment. If it is possible to record enough history of previous states and analyze a dynamic environment from history, then we can guess the next state of the environment. The forthcoming algorithms will still act myopically, but now on the estimated future state instead of the static current state.

Rather than present a general formulation, we will discuss a potential game approach to a cooperative moving target tracking problem.

3.2 Cooperative target tracking problem

The goal of a cooperative moving target tracking problem is to capture one moving target cooperatively. The target might be autonomous or controlled by a human. In this setting, the mobility of the target gives a dynamic environment from the perspective of the pursuing agents. We can expect that the target's next position is limited by its current position and velocity because it cannot exceed reasonable maximum speed and acceleration. In other words, the target's position is not completely random over the mission space. Therefore the proposed game model with appropriate prediction of the target's next position will show higher performance than the conventional one.

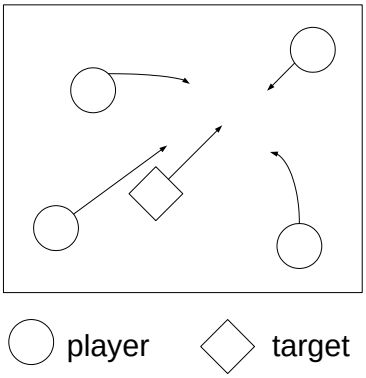


Figure 1: Moving target tracking problem

3.2.1 Modeling as a potential game

There is a player set $P = \{p_1, p_2, p_3, p_4\}$ and one target. Divide the mission space into a finite set of sectors denoted *S*. The action set of a player p_i is $A_i = S$ and the position of target is denoted by $b \in S$. For the sake of simplicity the target's position *b* is known to all players. (If the location of the target is unknown, then players need to explore the environment to discover the target, which requires different approach.) Each player p_i can communicate with a set of other players who are referred to as neighbors of p_i. The communication between players is bidirectional and limited to its neighbor set only. Each player p_i can cover the area centered at $a_i \in S$ with a radius r_p. The effective area to mark the target is centered at *b* and has a radius r_t. The joint coverage of sector *s* with action profile *a* is

$$J(s, a) = 1 - \prod_{p_i \in \mathrm{P}} [1 - f_i(s, a_i)] \tag{3.1}$$

where

$$f_i(s, a_i) = \begin{cases} 1 & if\ \|s - a_i\| < r_p \\ 0 & elsewhere \end{cases} . \tag{3.2}$$

We now define a function $T: S \to \mathbb{R}$ that determines the value of a sector *s*. The function *T* has larger values as the distance to the target is closer: $\|s_1 - b\| < \|s_2 - b\| \Leftrightarrow T(s_1) > T(s_2)$. However, even though a specific sector *s* is effective at time *t*, it might be

no longer effective at the next time step *t+1* since the target is moving. So if the trajectory of the target is predictable, the expected target's position can give a better result than current position when selecting the next action of the player. The defined function *T* is

$$T(s,t) = \begin{cases} A & if \ \|s - b_{exp}(t)\| < r_t \\ A \ exp(-\dfrac{\|s - b_{exp}(t)\| - r_t}{B}) & elsewhere \end{cases} \tag{3.3}$$

where *A* and *B* are constant values and $b_{exp}(t)$ is a expected target position at the next state of time *t*. The expected target position $b_{exp}(t)$ is calculated based on the player's belief about the target's behavior. We will assume that the target moves to the same direction and speed as the previous time step with a high probability. Therefore it is natural to expect that

$$\begin{aligned} b_{exp}(t) &= b(t) + \{b(t) - b(t-1)\} \\ &= 2b(t) - b(t-1). \end{aligned} \tag{3.4}$$

However if more precise expectation is available then $b_{exp}(t)$ can be an appropriate form reflecting the expectation rather than the equation (3.4).

In this setting, the global objective function takes on the form

$$\phi(a,t) = \sum_{s \in S} T(s,t) J(s,a). \tag{3.5}$$

The function $\phi(a,t)$ is maximized when the nearby area around the target is properly covered by players. In other words, players will surround the target as the objective function is maximized.

The utility of each agent is assigned a Wonderful Life Utility (WLU), i.e.,

$$\begin{aligned} U_i(a,t) &= \phi(a_i, a_{-i}, t) - \phi(a_i^0, a_{-i}, t) \\ &= \sum_{s \in S} T(s,t)(J(s, a_i, a_{-i}) - J(s, a_i^0, a_{-i})) \end{aligned} \tag{3.6}$$

where a_i^0 is the null action which means that player p_i does not exist. Therefore $\phi(a,t)$ and $U_i(a,t)$ are perfectly aligned, and the resulting game is a potential game.

3.2.2 Learning algorithm

Binary restricted log-linear learning explained in Section 2.3 was named binary since it chooses the next action between two options: the current action and one trial action based on the probability function (2.9) and (2.10). Therefore binary restricted log-linear learning consists of two steps: 1) selecting a trial action and 2) selecting a next action. The learning algorithm for the moving target problem is a modified version of binary restricted log-linear learning. We will call the modified version general restricted log-linear learning.

Like binary restricted log-linear learning, general restricted log-linear learning chose one player among the player set to select the next action among an available action set $R_i\big(a_i(t-1)\big) \subset A_i$. Let a_{ij} be the *j*-th element of $R_i(a_i(t-1))$. Then $a_i(t)$ is selected with the following probability:

$$Pr\big[a_i(t) = a_{ij}\big] = \frac{\exp\left\{\beta U_i\left(a_{ij}, a_{-i}(t-1)\right)\right\}}{D} \tag{3.7}$$

where $D = \sum_{a_{ij} \in R_i(a_i(t-1))} exp\left\{\beta U_i\left(a_{ij}, a_{-i}(t-1)\right)\right\}$ and $\beta \geq 0$ is an exploration parameter. By eliminating the binary trial action concept and comparing all available actions at the same time, a player has a greater chance to pick the best action. Accordingly, general restricted log-linear learning can maximize the potential function and utility function more quickly than binary restricted log-linear learning. On the other hand general restricted log-linear learning requires more calculations as $|R_i\big(a_i(t-1)\big)|$. increases.

3.3 Simulation

A simulation program is developed to show the performance in graphical and numerical manner. The program is developed using C++ and the Microsoft Foundation Class (MFC) library. For comparison learning without considering the dynamic characteristic of target is labeled '*static*' and learning as in Section 3.2 is labeled '*dynamic*'. The static case utilizes the current target position $b(t)$ instead of the expected position $b_{exp}(t)$ for the equation (3.3). On the other hand, the dynamic case follows the same algorithm described in Section 3.2.

To compare the performance of the two cases define new measurement $g(a,t)$ as follows:

$$g(a,t) = \sum_{s \in S} H(s,t) J(s,a) \tag{3.8}$$

where $H(s,t)$ is the target boundary check function defined as

$$H(s,t) = \begin{cases} 1 & if \ \|s-b\| < r_t \\ 0 & elsewhere \end{cases}, \tag{3.9}$$

and $J(s,a)$ is the same joint coverage function (13) from Section 3.2. The function $g(a,t)$ is the sum of the area around the target that is covered by players. Therefore a large g value means that the players' joint action profile surrounds the target effectively.

The performance ratio is calculated as

$$\frac{g(a_{dynamic}, t)}{g(a_{static}, t)} \tag{3.10}$$

where a_{static} and $a_{dynamic}$ are the joint action profile of static and dynamic learning respectively.

One trial of simulation is 1000 time steps. All players' positions are randomly initialized at the start of every trial. The mission space S is $\{1,2,\cdots,50\} \times \{1,2,\cdots,50\}$. Table 1 defines several numerical constants for the simulations.

Table 1: Constant values used in the simulation

r_t of the equation (3.3)	10
Player's coverage area r_p	5
A of the equation (3.3)	10
B of the equation (3.3)	10
Exploration parameter β	0.9

The moving target model is as follows. The target moves once every three time steps. Therefore the average speed of the target is 1/3 per time step. The default is that the target moves in the same direction as its previous movement. Assume that the target changes its direction every time step with probability $\boldsymbol{\sigma}$. Simulations were carried out with changing $\boldsymbol{\sigma}$ values from 5% to 40%. There were 100 trials were performed for one $\boldsymbol{\sigma}$. There are four pursuing players. The speed of target is selected considering that one of four agents is chosen to move one unit every time step. In other words, the average speed of each agent is 1/4 per time step. So the target's speed is slightly faster than the players.

Figure 2 shows that the performance ratio is decreased as $\boldsymbol{\sigma}$ increased because the dynamic case is based on the assumption that the target moves to the same direction as the previous time step with a high probability. Hence frequent change of direction makes it hard to predict the target's movement correctly.

Table 2 and Table 3 show numerical results of the simulation. In Table 2 it is observed that dynamic tracking always has better performance for all $\boldsymbol{\sigma}$ values than the setup in which the target's movement is not predicted. The covered area $g(a,t)$ around the target is getting larger as $\boldsymbol{\sigma}$ increased for both cases. It is because as $\boldsymbol{\sigma}$ is increased the target has more chance to change its direction and it makes players easy to catch the target's moving speed.

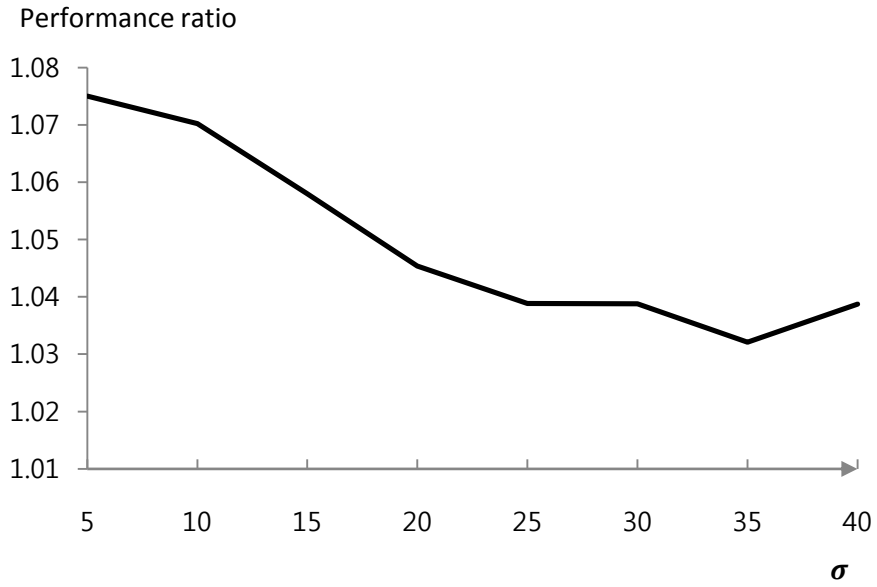


Figure 2: Relation between performance ratio and $\boldsymbol{\sigma}$

Table 2: Average performance of dynamic and static case with changing $\boldsymbol{\sigma}$

$\boldsymbol{\sigma}$	avr $g(a_{dynamic}, t)$	avr $g(a_{static}, t)$	Performance ratio
5%	51.10162	47.53675	1.074991875
10%	84.73768	79.17951	1.070197075
15%	100.83564	95.30884	1.057988325
20%	113.5538	108.6277	1.04534847
25%	125.60117	120.90271	1.038861495
30%	131.50041	126.59291	1.038765994
35%	136.40622	132.16771	1.032069179
40%	144.82695	139.42371	1.038754097

Table 3: Maximum and minimum performance result with changing $\boldsymbol{\sigma}$

$\boldsymbol{\sigma}$	$max\ g(a_{dynamic})$	max $g(a_{static})$	$min\ g(a_{dynamic})$	min $g(a_{static})$
5%	95.179	90.704	5.059	3.781
10%	130.32	115.635	43.08	43.866
15%	128.591	128.06	57.998	48.801
20%	141.599	140.083	77.671	67.819
25%	147.27	143.373	91.823	91.298
30%	159.177	154.155	104.812	100.462
35%	155.972	153.25	104.472	86.049
40%	164.067	162.477	118.671	115.624

3.4 Summary

This chapter presented the cooperative moving target tracking system. The global objective function and agents' utility functions were designed as a potential game, and the modified log-linear learning algorithm was applied. The algorithm utilized target's expected position instead of the actual position to adapt the target's dynamic characteristic. The simulation data shows that this approach improves the system's performance.

CHAPTER 4

DYNAMIC MAP COVERAGE PROBLEM

Consider a scenario that multi-mobile sensor system monitors a group of animals in a large area. Animals are continuously moving and their population distribution over the area is unknown. Each mobile sensor has limited sensor coverage and observed data is shared by all sensors. The objective is maximizing the number of animals under surveillance. In this scenario one can consider population density as event density, a group of mobile sensor as a player set, and an area in which mobile sensor can move as an action set for players. This chapter will illustrate this problem as a potential game and develop a suitable learning method.

4.1 Motivation

In the target tracking problem described in the previous chapter, the target location was known to all players so that the reward for every action over the map can be calculated even before a player takes a real action. In other words, the utility function over all actions was known. Therefore it was possible to develop a variant of log-linear learning which requires all utility values. This is a very strong assumption, and one cannot assume that utility values are computable in many practical problems.

This chapter will discuss a multi agent coverage problem without information about the event distribution over the coverage area. Since the event distribution is not given, players cannot utilize the same learning method of the previous chapter. We still assume that the other players' actions are still known, thus it is not the fully payoff based scenario presented in Section 2.3.2. Such a formulation in which players do not know

their utility functions and cannot measure other player actions will be discussed in Chapter 5.

4.2 Dynamic coverage as a potential game

The goal of the problem is to maximize coverage of events that drift slowly or jump to another location with some probability. The event value of a sector *s* at time *t* is $d(s,t)$. A player set $P = \{p_1, p_2, \cdots, p_n\}$ starts optimizing their positions over a 2-D sectored finite space S without any information about the initial event distribution $d_0(s) = d(s,0)$ for all $s \in S$. An action set A_i for player p_i is also S, i.e. $a_i \in A_i = S$. At every time step *t*, player p_i can measure the event distribution $d(s,t)$ within its sensor coverage radius r_i. Let a set of sectors within the coverage of a player p_i at time t be $S_{i,t}$. Then

$$S_{i,t} = \{s \in S : \|s - a_i(t)\| \leq r_i\}. \tag{4.1}$$

Data sampled in the past does not represent the current event density of the location since event density is changing over time. Moreover it is assumed that there is zero mean observation error of event density, so observed data is not 100% reliable. All players share their observed data and actions. As in the previous chapter, a player's available action set for the next time step is restricted by its current action.

Define a global objective function $\phi(a_i, a_{-i}, t)$ as the sum of event density in the area covered by joint action *a*, i.e.

$$\phi(a_i, a_{-i}, t) = \sum_{s \in S_t^*} d(s,t) \tag{4.2}$$

where $S_t^*(t) = \bigcup_{i=1}^{n} S_{i,t}$. Define an individual utility function $U_i(a_i, a_{-i}, t)$ for a player p_i using Wonderful Life Utility as

$$\begin{aligned} U_i(a_i, a_{-i}, t) &= \phi(a_i, a_{-i}, t) - \phi(a_i^0, a_{-i}, t) \\ &= \sum_{s \in (S_{i,t} \setminus S_{-i,t})} d(s,t) \end{aligned} \tag{4.3}$$

where a_i^0 is a null action and $S_{-i,t}$ is an area covered by a_{-i} at time t. Consequently it is a potential game with potential function ϕ and utility U_i for a fixed time *t*.

Recall that players know observed data only instead of actual event density $d(s,t)$, so players cannot calculate U_i directly in this scenario. Instead players will utilize an estimated utility $\widehat{U}_i$ which depends on estimated event density $\hat{d}(s,t)$.

Estimated event density $\hat{d}$ is initialized at time $t = 0$ with arbitrary offset value $\hat{d}_0$. Let $o(s,t)$ be the observed density of place *s* at time *t*, i.e.

$$o(s,t) = d(s,t) + v \tag{4.4}$$

where $s \in S_t^*$ and v is an independent and identically distributed random variable with zero mean. In other words, $o(s,t)$ includes zero mean observation noise. Then updating rule for $\hat{d}(s,t)$ is

$$\hat{d}(s,t) = \begin{cases} \hat{d}(s,t-1) + \rho_1 \left(o(s,t) - \hat{d}(s,t-1)\right), & \text{if } s \in S_t^* \\ \hat{d}(s,t-1) + \rho_2 \left(\hat{d}_0 - \hat{d}(s,t-1)\right), & \text{otherwise} \end{cases} \tag{4.5}$$

Where ρ_1 and ρ_2 are arbitrary constants such that $0 < \rho_1 \leq 1$ and $0 < \rho_2 \leq 1$. This updating rule only depends on $\hat{d}(s,t-1)$ and $o(s,t)$, so the required memory has finite length. If the observed density at place *s* is not reported for a long time, the estimated density value will converge to $\hat{d}_0$. This characteristic will be referred as fading memory concept.

Then estimated utility $\widehat{U}_i(a_i, a_{-i}, t)$ is defined as follows:

$$\widehat{U}_i(a_i, a_{-i}, t) = \sum_{s \in (S_{i,t} - S_{-i,t})} \hat{d}(s,t). \tag{4.6}$$

Hence $\widehat{U}_i(a_i, a_{-i}, t)$ is the sum of the estimated density function in places exclusively covered by a player p_i.

4.3 Learning algorithm

Since the event density over the given space is now unknown, players have two objectives in their learning: exploration and exploitation. Exploration is sampling unknown places including not only places never visited before but also places visited. Exploitation is optimizing players' formation to maximizing a global objective. Considering equation (4.5), if the observed value of a place is lower than $\hat{d}_0$ then a player will prefer exploring a different unknown area, which has utility value $\hat{d}_0$. Hence one can consider $\hat{d}_0$ as a threshold between exploration and exploitation. If $\hat{d}_0$ is relatively high then players have more incentive to explore and vice versa.

The learning algorithm for this problem is similar to binary restricted log-linear learning in Chapter 2 except it will utilize the estimated utility instead of actual utility. In original log-linear learning only one player can change its action, but in the learning algorithm in this chapter all players are allowed to change their action every time step. It allows players to explore more places in a short time and update estimated density effectively.

Every time step all players choose a trial action $\hat{a}_i$ among its current available action set $R_i\big(a_i(t-1)\big) \subset A_i$ with the following probability:

$$Pr\big[\hat{a}_i(t) = a_{ij}\big] = \frac{\exp\left\{\beta \widehat{U}_i\left(a_{ij}, a_{-i}(t-1)\right)\right\}}{D} \tag{4.7}$$

where $D = \sum_{a_{ij} \in R_i(a_i(t-1))} exp\left\{\beta \widehat{U}_i\left(a_{ij}, a_{-i}(t-1)\right)\right\}$ and $\beta \geq 0$ is an exploration parameter. Players choose their next action as follows

$$a_i(t) = \begin{cases} \hat{a}_i(t), & \text{with probablity } (1-\omega) \\ a_i(t-1), & \text{with probability } \omega \end{cases} \tag{4.8}$$

where ω is an arbitrary inertia parameter with $0 \leq \omega \leq 1$.

After the playing chosen action, every player $p_i \in P$ observes the event density in their coverage $S_{i,t}$ and updates the estimated density function $\hat{d}(s,t)$ for all $s \in S$ using

equation (4.5). Since observed data is shared by all players, players have the same estimated utility over the space *S*.

4.4 Simulation

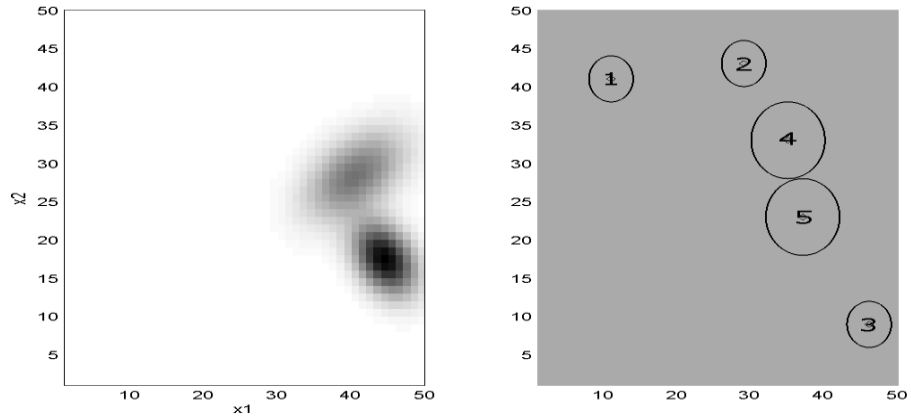
The performance of the designed system is simulated using MATLAB. A mission space for the simulation is a set of sectors, $S = \{1,2,\cdots,50\} \times \{1,2,\cdots,50\}$. There is a player set $P = \{p_1, p_2, p_3, p_4, p_5\}$ and all players can be at any sector over S. Players have limited moving speed so that the available action set for every time step is also limited. All players can move to only adjacent sectors from the current sector in one time step. It is assumed that three players have sensor coverage with radius 3 and the other two players have coverage radius 5. In other words, the coverage radius is defined as $r_1 = r_2 = r_3 = 3$ and $r_4 = r_5 = 5$. There exists event density function $d(s,t) \geq 0$ for $\forall s \in S, t \geq 0$ and $d(s,t)$ is sum of two 2-D Gaussian probability density functions. Let the two Gaussian probability density functions be $N_1(\mu_1(t), \Sigma_1)$ and $N_2(\mu_2(t), \Sigma_2)$. The mean vector of the first density function, $\mu_1(t)$, is drifting slowly with random direction and the mean of another density function, $\mu_2(t)$, jumps to a random sector s ∈ S with a probability 0.03. Discount ratios for the estimated utility function (4.5) are selected as $\rho_1 = 0.5$ and $\rho_2 = 0.95$.

Figures 3~5 are the results of the simulation. The left screen of the simulation shows the actual event distribution $d(s,t)$ which is unknown to players. The right screen illustrates the estimated event distribution $\hat{d}(s,t)$ and positions of players. In this simulation, the mean value of the jumping event density $N_2(\mu_2(t), \Sigma_2)$ changes its location twice at time step 60 and 140. The mean of another event density function $N_1(\mu_1(t), \Sigma_1)$ moves slowly during the entire simulation. At time $t = 0$, $\hat{d}(s,t)$ for all $s \in S$ is initialized with same offset value. As time increases, the estimated density of a place previously visited is converging to the offset value. This feature is illustrated as

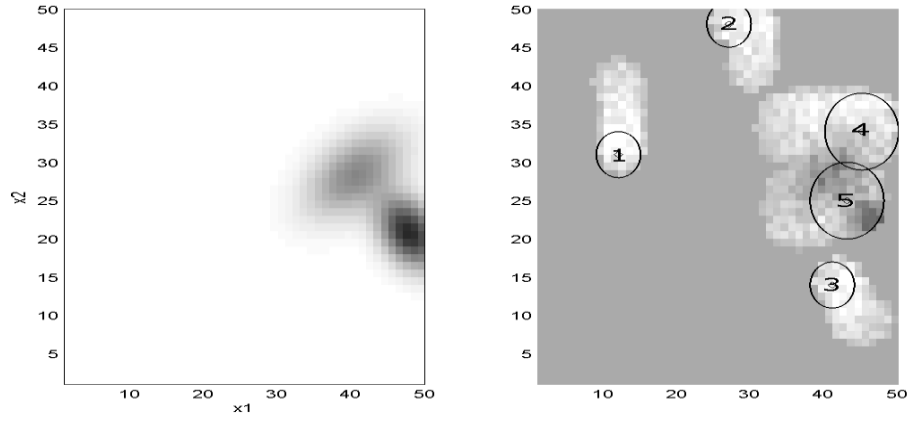
gradually blurry trail behind players. A player p_5 tagged with number 5 in the figures keeps following the drifting event after time step 40. All players except player p_5 continue exploration until players p_3 and p_4 encounter an event area at time step 140. At time step 160, players p_2, p_3, p_4 and p_5 successfully covered most of events and player p_1 is exploring unknown places.

4.5 Summary

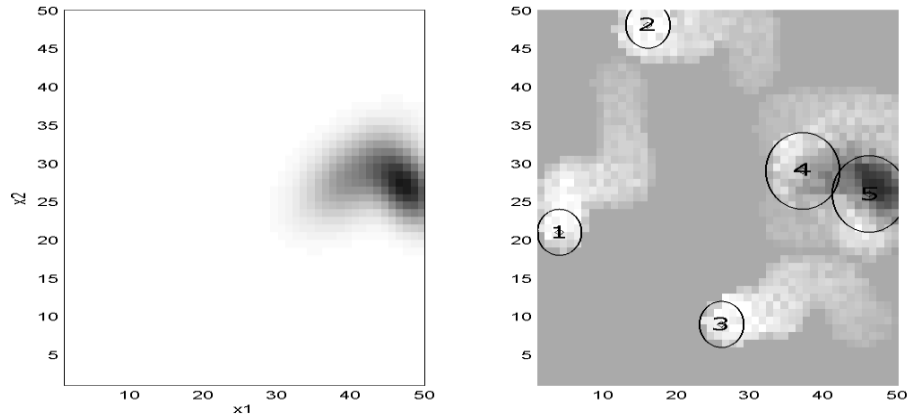
We formulated a multi agent dynamic map coverage system with an assumption that the event distribution is unknown to players. The global objective function and agents' utility functions form a potential game. The estimated utility approximates the actual utility based on players' observation, and the fading memory concept is applied to the estimated utility updating rule to adapt the time variant environment. Each player selects its action using the modified log-linear learning with inertia. The simulation result shows that this system successfully covers dynamic events.



(a) $t = 0$

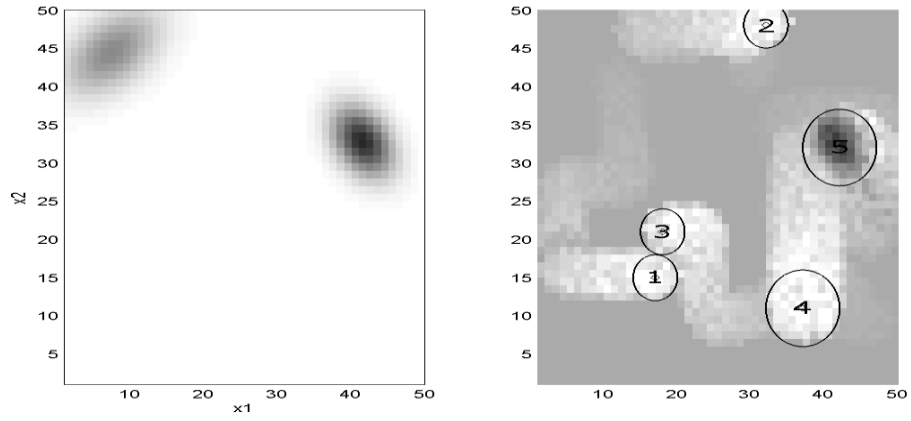


(b) $t = 20$

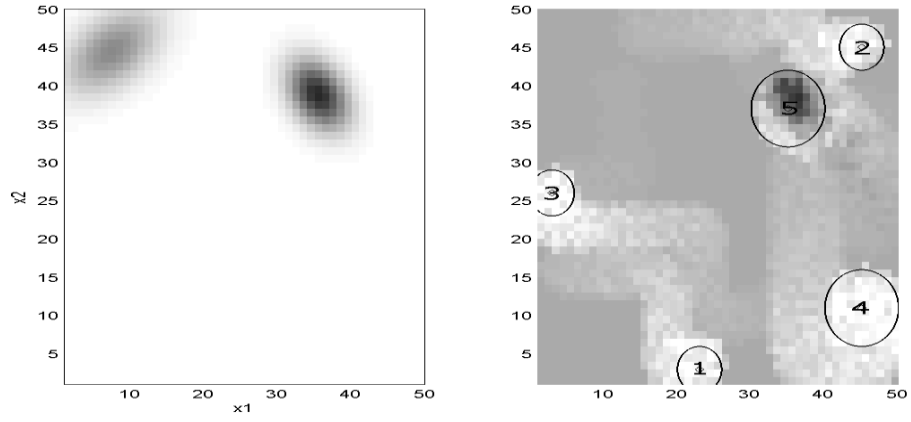


(c) $t = 40$

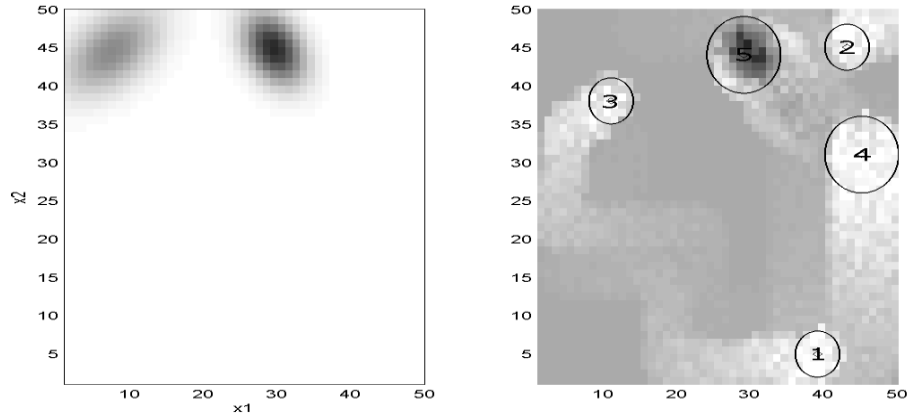
Figure 3. Simulation results of dynamic map coverage at time step 0, 20 and 40. The left window of each time step shows actual event density, and the right window shows estimated density and players' coverage. Darker color has larger value in both screens. Each player's coverage is tagged with number for identification.



(a) $t = 60$

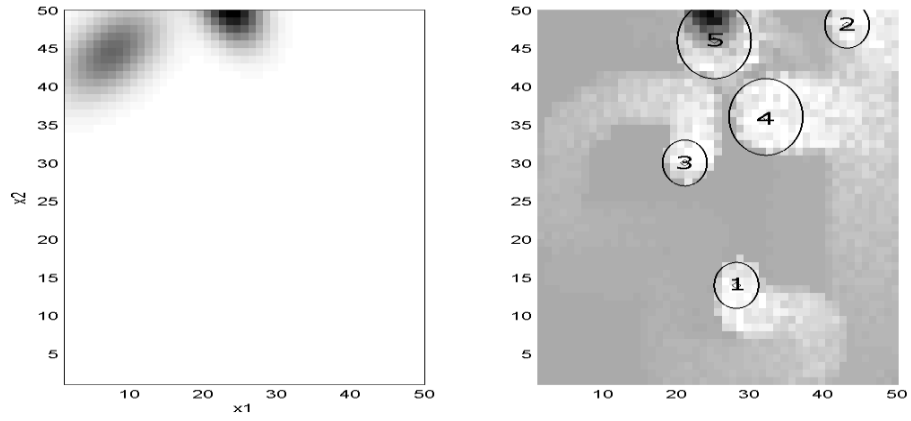


(b) $t = 80$

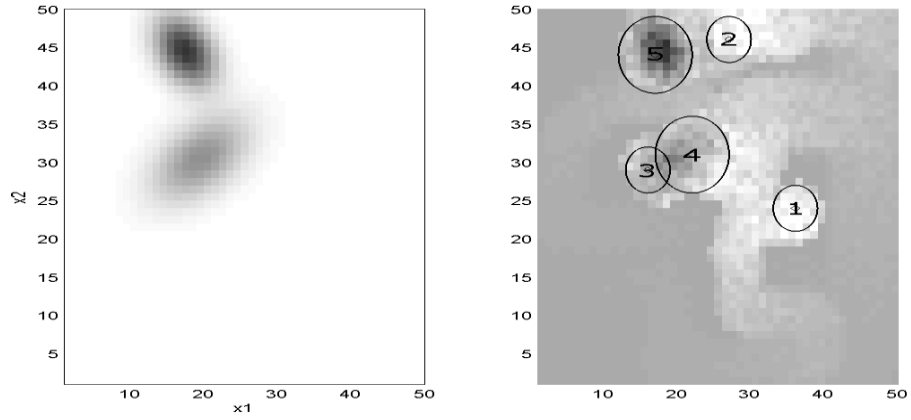


(c) $t = 100$

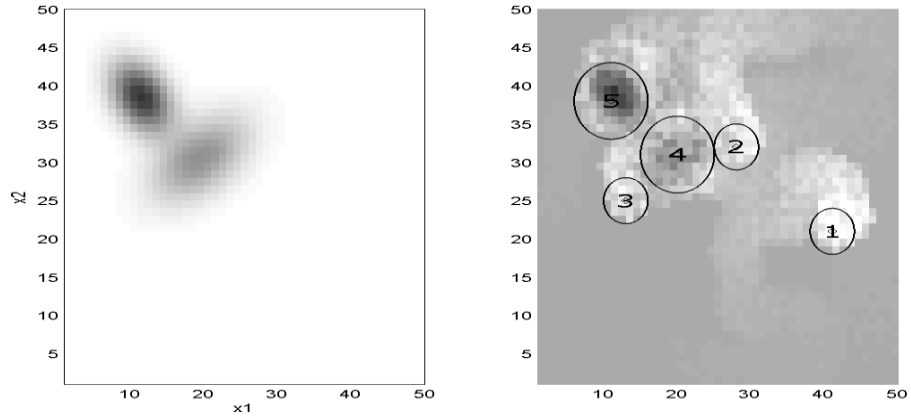
Figure 4. Simulation results of dynamic map coverage at time step 60, 80 and 100. The left window of each time step shows actual event density, and the right window shows estimated density and players' coverage. Darker color has larger value in both screens. Each player's coverage is tagged by number for identification.



(a) $t = 120$



(b) $t = 140$



(c) $t = 160$

Figure 5. Simulation results of dynamic map coverage at time step 120, 140 and 160. The left window of each time step shows actual event density, and the right window shows estimated density and players' coverage. Darker color has larger value in both screens. Each player's coverage is tagged by number for identification.

CHAPTER 5

PATH OPTIMIZATION PROBLEM

Consider the scenario that multiple cleaning service robots in a park try to empty trash cans on their route. An empty trash can becomes full with certain probability every day, and once it is filled with garbage then it remains in full status until any cleaning robot empty it. Every cleaning robot knows its own route and the number of trash cans it has emptied, but it does not have any information about other robots' routes or work results. How can the group of robots form efficient routes to minimize the average number of full trash can in the park? This chapter will represent this scenario as a multi-agent problem interpreting a cleaning robot as a player, route of a robot as a player's action and a trash can as an service location with status value.

5.1 Motivation

This chapter will consider a more challenging problem: payoff based potential games in dynamic environment. As explained in Section 2.3.2, payoff based dynamics only depend on the player's own realized payoffs. Therefore a player cannot observe the other players' actions or payoffs. This is a severe restriction considering that one player's utility in multi-agent system depends on other players' actions. Even if playing a specific action gives higher payoff than other actions it does not guarantee that the action is better since the higher reward might be from other players' actions. However in many settings of multi-agent systems, agents' knowledge about their environment and communication between agents can be limited or even prohibited. In that case payoff based dynamics

offer an alternative since they do not require any information but a player's own realized payoffs.

Reference [11] introduced payoff based dynamics for weakly acyclic games which are a more general case of potential games, and a simple congestion game was described as an illustrative example. This chapter will expand one of payoff dynamics introduced in [11] to a dynamic and stochastic environment with a large action set

5.2 Path optimization as a potential game

The objective of this chapter's problem is minimizing the number of service location with a certain status. A total of *m* service locations are distributed over a 2-D mission space $S = X \times Y$. Let the service location set be $E = \{e_1, e_2, \cdots e_m\}$. Each service location e_j is a tuple containing two states: status $q_j \in \{0,1\}$ and location $l_j \in S$. Each service location $e_j \in E$ is initialized to status $q_j = 1$ and arbitrary location l_j. A player p_i in a player set $P = \{p_1, p_2, \cdots, p_n\}$ has an action set A_i and each action $a_i \in A_i$ is a parameter vector that describes player's route. For convenience, we will limit the shape of a player's route to a circle. Then an action a_i defines a specific circular route of a player p_i. It is a tuple (x_i, y_i, r_i) where $x_i \in X$ and $y_i \in Y$ are the coordinates of the center, and $r_i \in R$ is the radius. Consequently an action set A_i is a subset of $\mathbb{R}^3$. A player p_i moves along its route represented by its action vector a_i with fixed speed. All service locations within a certain distance d_i from any player p_i are forced to change their status to 0. Every time step, service location e_j that is not influenced by a player and has status $q_j = 0$ changes its status to 1 with a probability k.

Define a potential function $\phi(a_i, a_{-i})$ as follows:

$$\phi(a_i, a_{-i}) = \lim_{t \to \infty} \frac{\sum_{\tau=1}^{t} E\{v(a_i, a_{-i}, \tau)\}}{t} \tag{5.1}$$

where $\nu(a_i, a_{-i}, \tau)$ is number of service locations with status 0 at time τ when every player p_i starts rotating a route a_i from $(x_i + r_i, y_i)$ at time 0. Hence $\phi(a_i, a_{-i})$ is an average value of expected number of service locations with status 0 over time. Using Wonderful Life Utility with $\phi(a_i, a_{-i})$, a player's utility function is defined as follows:

$$U_i(a_i, a_{-i}) = \phi(a_i, a_{-i}) - \phi(a_i^0, a_{-i}). \quad (5.2)$$

Consequently this forms a potential game. Loosely speaking, $U_i(a_i, a_{-i})$ is the marginal contribution of a player p_i to increase number of service locations with status 0. In other words, it is the number of service locations whose status is switched from 1 to 0 by a player p_i.

5.3 Learning algorithm

The learning algorithm for the path optimization problem is based on Sample Experimentation Dynamics from [11] and presented in Chapter 2. Let an exploration phase consist of *m* action trials. One action trial means turning a complete circle with the center and radius specified by the action. Since a player's moving speed is fixed and a circumference of a circle is proportional to its radius, the time for one action trial can be varied. Consequently a players' exploration phase is not synchronized even though the number of action trials in one phase is the same for all players. Index an action a_i played in the t_2-th trial of the t_1-th exploration phase as $a_i(t_1, t_2) = a_i(mt_1 + t_2)$ where t_1 is the exploration phase time and t_2 is the exploration action time. Note that $(mt_1 + t_2)$ indexes the order of an action trial and it does not indicate a time step that the action is made. Therefore $a_i(t_1, t_2)$ and $a_j(t_1, t_2)$ with $i \neq j$ may not be executed simultaneously even though the indices are the same. The baseline action is indexed with exploration phase time as $a_i^b(t_1)$. By construction, exploration phase time and exploration action time satisfy $t_1 \geq 1$ and $m \geq t_2 \geq 1$.

The learning algorithm is as follows. First, each player p_i chooses a random initial baseline action $a_i^b(1) \in A_i$, then each player starts an exploration phase. During the exploration phase a player plays m trials and each trial action is chosen as its baseline action with probability $(1-\varepsilon)$ or a new random action other than the baseline action with probability ε. The variable ε will be referred to as the exploration rate.

Every time step, a player p_i moves along its route based on its current trial action with fixed speed and checks whether there is an service location whose status is 1 within a distance d. If there are service locations satisfying these conditions, then the status values of the service locations are changed to 0. Define an observed payoff $O_i(mt_1 + t_2)$ to be the number of service locations whose status are changed by a player p_i during t_2-th trial of t_1-th exploration phase.

After playing a trial action, a player p_i updates its average payoff function as follows:

$$\hat{V}_i(a_i, t) = \begin{cases} \hat{V}_i(a_i, t-1) + \rho\left(O_i(t) - \hat{V}_i(a_i, t-1)\right), & if\ a_i \text{ was played at } t \\ \hat{V}_i(a_i, t-1), & \text{otherwise} \end{cases} \tag{5.3}$$

where $t = mt_1 + t_2$. This update rule adapts a fading memory concept which is used for the estimated event density update rule in Chapter 4.2.1. The constant ρ is a update weight satisfying $0 < \rho \leq 1$ and it determines the influence of old payoff history. Unlike the original Sample Experimentation dynamics in [11], this average payoff function is never reset at the end of exploration phase. There are two reasons why a notion of fading memory is applied instead of resetting average payoff every exploration phase.

The first reason is the players' large action set. The congestion game in [11] has a relatively small action set which consists of several choices. On the other hand an action set in this path optimization problem is in $\mathbb{R}^3$. Resetting a player's average payoff

requires too long exploration of an phase to explore the action set sufficiently, and as a result it will slow the entire system's learning speed.

Another reason is that player exploration phases are not synchronized in this problem. Even if a player starts a new phase, the other players may stay in the same exploration phase as the previous time step. Consequently recent payoff history is not useless for a new exploration phase and a player does not have to reset its average payoff.

After completing an exploration phase, a player updates its baseline strategy. Define a better payoff set $A_i^*(t)$ as follows

$$A_i^*(t_1) = \{a_i \in A_i : \hat{V}_i(a_i, mt_1 + m) > \hat{V}_i(a_i^b, mt_1 + m)\}. \quad (5.4)$$

Note that the time index for $\hat{V}_i$ in equation (5.4) is $(mt_1 + m)$ because A_i^* is calculated at the end of an exploration phase. Then each player updates its baseline action with following rules:

- If $A_i^*(t_1)$ is empty, then $a_i^b(t_1 + 1) = a_i^b(t_1)$
- If $A_i^*(t_1)$ is not empty, then
  - $a_i^b(t_1 + 1) = a_i^b(t_1)$ with probability ω
  - $a_i^b(t_1 + 1)$ is chosen randomly with uniform distribution over $A_i^*(t_1)$ with probability $(1 - \omega)$

The term ω will be referred to as the player's inertia. Lastly, a player initiates a new exploration phase with the updated baseline action and repeats these 'exploration phase' and 'baseline update' processes.

5.4 Simulation

To illustrate performance of the path optimization system, a simulation program was developed using MATLAB. The size of the player set in the simulation is 5 and all players' moving speed is 1.6 per time step. There are a total of 100 service locations on

the field. Every time step, each service location with status 0 changes its status to 1 with probability 0.04.

An action set A_i is a Cartesian product of a radius, center's x-coordinate and y-coordinate. The actual action set used in a simulation is as follows.

$$A_i = X \times Y \times R = \{0,5,10,15, \dots,50,55\} \times \{0,5,10,15, \dots,50,55\} \times \{10,15,20\}$$

Therefore size of the action set is $12 \times 12 \times 3 = 432$. The reason to discretize an action space at intervals of 5 is to reduce the size of the action set sufficiently.

Constant values for the algorithm used in the simulation are shown in the table 4.

Table 4: Constant values used in the simulation

inertia ω	0.4
exploration rate ε	0.03
update weight ρ	0.5

The number of trials per exploration phase is 100 and the simulation was terminated when at least 30 exploration phases are performed for all players. Note that players are not synchronized with each other and time for a trial depends on the radius of a player's path. So at the end of simulation, players can be in a different exploration phase from others.

The time for running one action depends on the radius of the path defined by the action. Considering that players' moving speed is 1.6 per time step, the time steps required for one action trial, i.e. turning a complete circle route, is 40 steps for a route with radius 10, 59 steps for radius 15, and 79 steps for radius 20.

Figure 6 shows the optimized paths for multiple agents. The x-axis in Figure 7 is time and the y-axis is total number of events spots with status 1. Note that the x-axis has a much larger value than the exploration phase or trial counts since one action trial

requires more than 40 time steps. The objective of this system is to minimize the number of active station. After optimization, the number of active station is staying between 50~55 as shown in Figure 7.

The important point in this simulation result is not the literal value 50, but the fact that the number of active service location is decreasing over time. Because the result value 50 can be relatively small or large according to various conditions such as the event changing rate or player's moving speed. The main result of this graph is that the payoff based algorithm is adapting in a stochastic environment successfully.

5.5 Summary

In this chapter, payoff based path optimization in a statistical environment was presented. Any communication between the players is not allowed so that each player knows its realized action and reward history only. The global objective function and agents' utility functions are designed to build a potential game. The modified Sample Experiment Dynamics algorithm was suggested as a learning algorithm. Average payoff function in the algorithm estimates expected reward for players' actions. The simulation data shows that this system effectively adjusts players' routes without communication between the players.

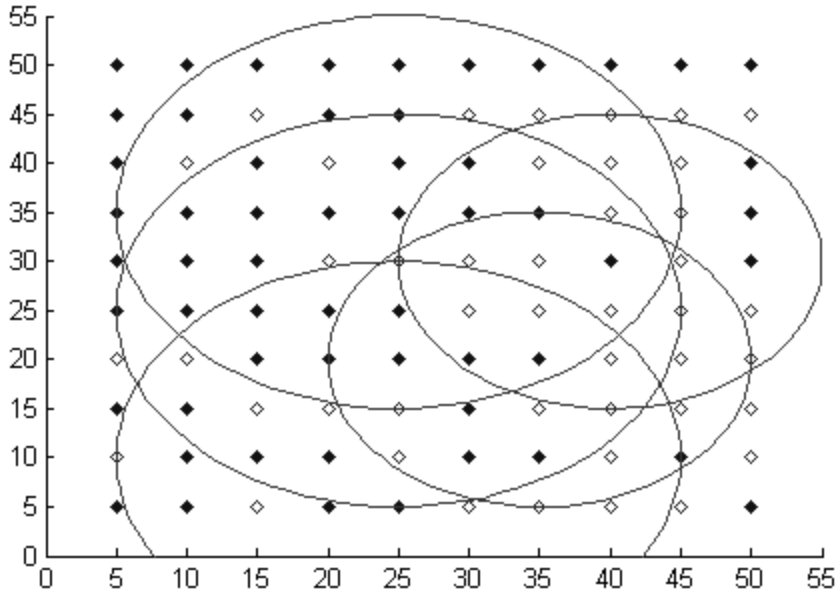


Figure 6. Optimized routes of players. Large circles show the players' baseline actions at the end of the simulation. Empty small spots are service locations with status 0 and filled spots are service locations with status 1 at the last time step.

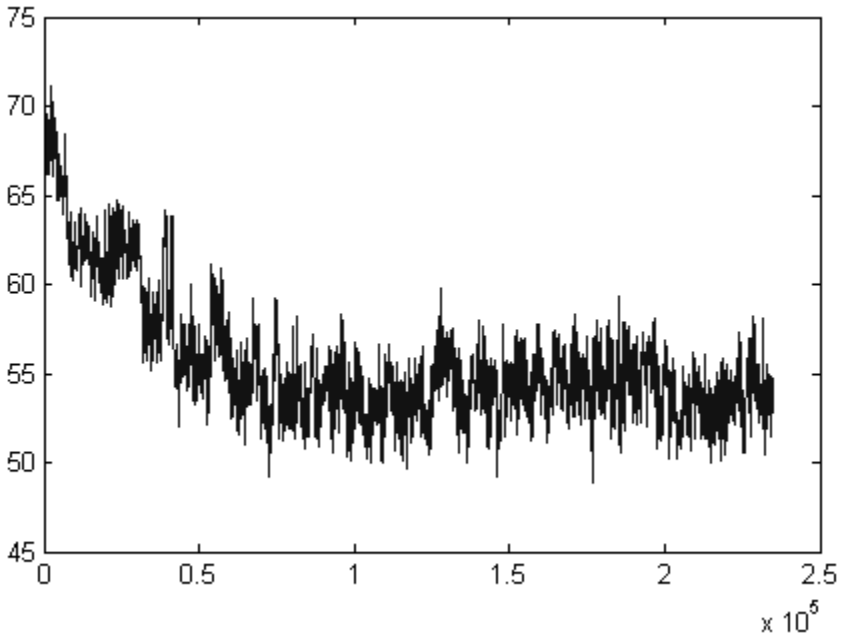


Figure 7. Average number of service locations with status 1 over time. The x-axis is time step and the y-axis is average number of service locations with status 1 during last 300 time steps.

CHAPTER 6

TESTBED IMPLEMENTATION

In the previous chapters, the performance of proposed potential game based cooperative control is demonstrated in simulation. This chapter presents a test bed implementation of the target tracking problem. The test bed both demonstrates the approach and highlights various implementation issues.

6.1 Experiment devices

6.3.1 Khepera III

Figure 8: Khepera III mobile robot

Khepera III [16] is a mobile robot model developed by K-team Corporation (see Figure 8). It is driven by two wheels, and multiple sensors on its base make possible long range and short range object detection. The robot base uses the KoreBot computer board which is designed for robotic application. The KoreBot board features an embedded Linux operating system and standard compact flash extension cards supporting WiFi, Bluetooth and many others. The size of the Khepera III is 130 mm x 70 mm and its maximum running speed is 0.5m/s. The KoreBot board has a 400 MHz ARM

microprocessor and 64MB ram. The on-board Linux system for the KoreBot board is version 2.6. An application programming interface (API) for application level development is provided by the manufacturer. A 802.11b wireless network extension card is used for the experiment

6.1.2 Vicon motion capture system as GPS

Figure 9: The Vicon motion capture system in action [17].

The purpose of the motion capture system is to record movement of an given object or group of objects and analyze the data to form a digital model (see Figure 9). Motion capture systems are used in 3D animation and filmmaking, sports, and medical application. The Vicon MX [17] which is used in this experiment is a motion capture system developed by Vicon Corporation. The Vicon MX collects 2D data from 8 specialized cameras and reconstructs it to 3D data. The specialized camera system of the Vicon detects infrared light reflected by markers attached to the objects. Unique geometric patterns of the markers make the system distinguish one object from others. Using these features of the Vicon, reference [18] developed a Global Positioning System (GPS) lab to track the movement of mobile agents. The GPS can measure the robots' global positions and broadcast data through Bluetooth communication in real-time. See [18] for a more detailed discussion.

6.3. Experimental system

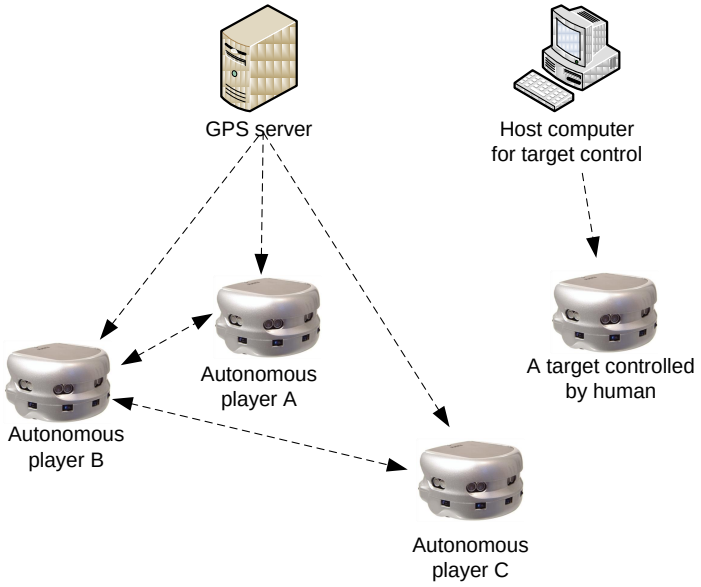


Figure 10: Structure of experiment system. The dashed line shows communication between elements. The figure illustrates the situation when the player B is a common neighbor of the player A and C.

Robot ID	X coordinate	Y coordinate	orientation

Figure 11: GPS protocol

The experiment system consists of mobile robots, the GPS system, and a host computer (See Figure 10). There are four mobile robots: three autonomous robots as players in the target tracking problem and one human controlled robot as a moving target. The three player robots make decisions in a distributed manner using local information.

The GPS checks all mobile robots' locations and broadcasts the information periodically. The GPS protocol has a structure in Figure 11. Even though GPS can calculate the 3D position of the robots, the mission space in the target tracking is 2D so Z-axis information is omitted in the protocol. If a player can access each other player's location information, then all players share global information which is not suitable for

the experiment's purpose to test a distributed system. Hence to make a distributed system, the receivers, i.e. mobile robots, only access the data tagged with its ID and ignore any other players' information in the GPS protocol. Therefore the coordinate of other robots should be obtained from peer-to-peer communication between robots. However the location of the target robot can be accessed by any player robot.

A host computer is required for the target robot control and it is independent from the cooperative control algorithm. Commands for the target robot are received through a host computer and sent to the target robot via Bluetooth. The host computer user can use a keyboard to move the target with constant speed and four directions (forward/ backward/ left turn/ right turn). Details of the developed application in the host computer will be discussed in Section 6.2.2.

6.2.1 Network

Generally there are two types of wireless network architectures: centralized and decentralized. A centralized network has at least one intermediate device such as a router or access point which manages a routing table and relays data between agents. On the other hand, a decentralized network does not have any intermediate device and each agent is responsible for their communication. For our purposes, a decentralized network is the appropriate choice than centralized network for the experiment. The network connection between robots is based on a wireless ad-hoc network which is one tyoe of distributed network. In a wireless ad-hoc network, each node participates in routing by forwarding data for other nodes, and so the decision of which nodes to forward data is made based on the network connectivity. The players in the moving target tracking problem can communicate with its neighbor only so actually it does not require data routing; all communication is one hop peer-to-peer communication.

The network protocol for communication between players is User Datagram Protocol (UDP). UDP was designed by David P. Reed in 1980 and formally defined in

RFC 768. UDP provides an unreliable service and assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system [19]. In the multi-agent experiment system, the network is required to update GPS information and neighbors' states in real-time. So even though UDP cannot provide guaranteed delivery, it is a reasonable choice due to its simplicity and low overhead.

6.2.2 Mobile robot remote control program

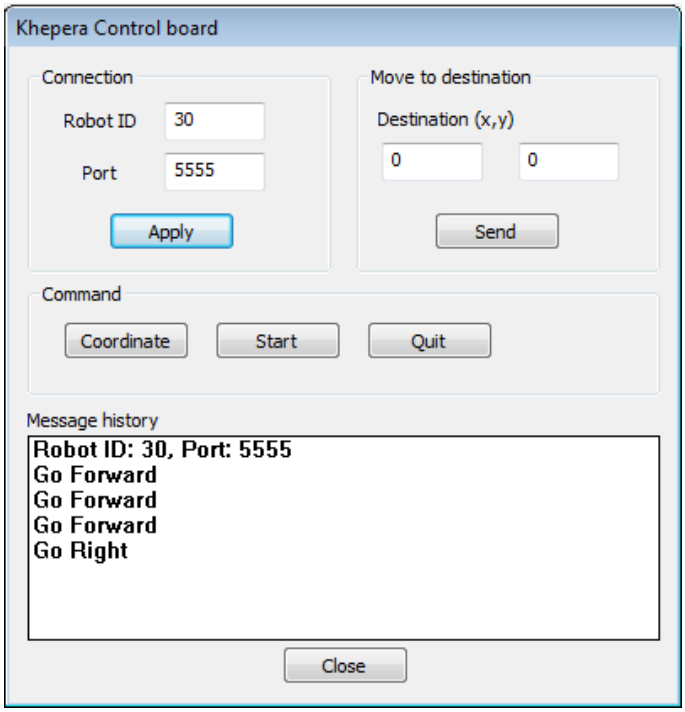


Figure 12: Mobile robot remote control program interface

The mobile robot remote control program (see Figure 12) was developed to manage the experiment system effectively. The major function of the program is to send pre-defined commands to the desired robot. It can be used for controlling the mobile robots' motion and check their current status. In the experiment the moving target is

human-controlled using this program. The program is developed using C++ and the Microsoft Foundation Class (MFC) library. All command protocols should be consistent with the embedded program of the mobile robot. The program consists of 4 major functions as follows:

1. Select robots and network port for remote control

2. Control the selected robot manually

3. Start or stop the multi-agent system algorithm

4. Show remote command history

In the connection section, a user can select robots using ID and network port for communication. A user can select more than one robot ID by typing multiple IDs separated by commas. Then all commands will be sent to multiple robots at the same time. This function is designed to manage the multiple mobile robot system.

There are two ways to manually control robot's motion: keyboard control or destination selection. A user can move the selected robot to forward/ backward/ right turn /left turn (from robot's point of view) using arrow buttons on a keyboard. Or some specific coordinate in (x, y) format can be set to move the robot.

The third part of the program is managing automatic control which is the cooperative control algorithm for players in this experiment. The remote control program can only turn on or off the automatic algorithm, and all algorithms must be stored in the robot's individual memory. This automatic control management can be used for starting or stopping programs in player robots. When the cooperative control program is active, any manual command such as keyboard control does not work. For debugging purpose a 'coordinate' button is added to the program to print robot's current coordinate in (x, y) format.

Lastly, the bottom of the program shows the history of commands by a user. All communication between the program and robots uses the aforementioned User Datagram Protocol (UDP).

6.2.3 Path planning

How to navigate a mobile robot from current position to the desired position is another issue for the experiment. There are some well-known path planning methods such as Voronoi diagram [20] and A* (pronounced "A-star") algorithm [21]. However path planning is not a major topic for this research, so a simpler path planning algorithm is applied to the experiment.

First, a robot decides its next position using the cooperative control algorithm. After choosing a goal, it will calculate distance and angle $\boldsymbol{\theta}$ from current location using GPS information. The robot turns its body with the calculated angle $\boldsymbol{\theta}$ to face the goal location. Then the robot goes straight to the goal position while continuously checking angular error. The major causes of angular error are GPS observational error and motor control error. If the angular error becomes larger than pre-defined threshold, it will stop and repeat angle correction process again. Lastly, if the robot arrives in an area close enough to the goal it finishes path planning.

This approach is very straightforward to implement and requires little calculation and system memory. However it may take much longer time to navigate than other path planning methods.

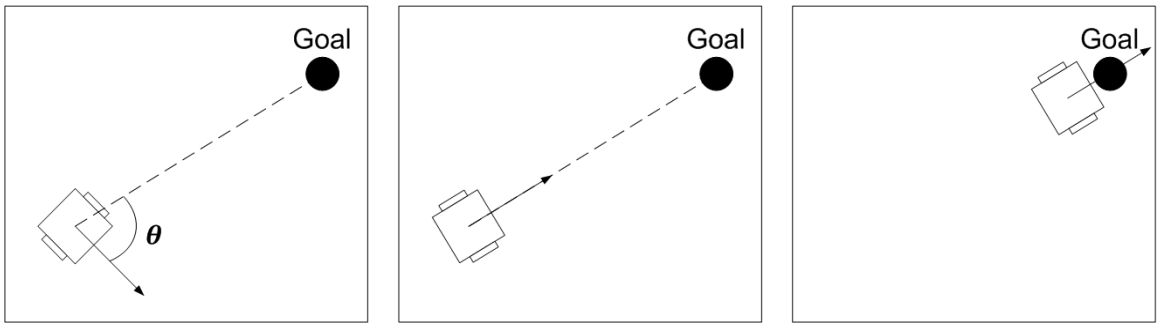


Figure 13: Robot calculates angle $\boldsymbol{\theta}$ to the goal in phase 1 and turns its direction in phase 2. In phase 3, robot finishes path planning if it arrives in a certain range of the goal.

6.3 Consideration for the experiment

6.3.1 Modified restrictive log-linear learning

Restrictive log-linear learning was applied to the target tracking problem in Chapter 3. Restrictive log-linear learning allows only one player to move at each time step, but some global synchronization is needed to realize it in the experimental system. For synchronization, a central control unit which can always communicate with all players is required, which is not desirable for a multi-agent system. As a result modified a restrictive log-linear learning is developed to solve this problem.

In modified restrictive log-linear learning, all players are allowed to change their actions every time step. However it does not mean that players actually change their action every time step. A notion of *inertia* in Sample experimentation dynamics is applied instead of strictly synchronizing players' action. Every time step, each player p_i randomly selects a temporary action $\hat{a}_i(t)$ from its available action set $R_i(a_i(t-1))$ with the following probability:

$$\Pr\left[\hat{a}_i(t) = a_i\right] = \frac{exp\{\beta U_i(a_i, a_{-i}(t-1))\}}{\sum_{\bar{a}_i \in R_i(a_i(t-1))} exp\{\beta U_i(\bar{a}_i, a_{-i}(t-1))\}} \tag{6.1}$$

for $\forall a_i \in R_i(a_i(t-1))$ and an arbitrary parameter $\beta \geq 0$. Then p_i decides its action for time t as follows:

$$\Pr[a_i(t) = \hat{a}_i(t)] = 1 - \omega \tag{6.2}$$

$$\Pr[a_i(t) = a_i(t-1)] = \omega. \tag{6.3}$$

The arbitrary variable $\omega \in (0,1)$ is inertia of this learning method. If ω is large enough, a set of players can stochastically approximate original log-linear learning's feature without global synchronization.

6.3.2 Physical limitation of mobile robots

The Khepera III was used as the mobile agents in this research. Since the maximum speed and acceleration of the Khepera III is limited, the available action set for each player is also limited. In the simulation, it was assumed that the player can move to an adjacent position with the same speed, but it might not be feasible in the experiment because of time for turning heading direction. For example, moving to the location in front of the player is faster than moving to the location behind the player since it requires time for turning the robot's body.

6.4 Results

It is observed that three player robots approach to the target robot as already shown in the simulation result. However the system's convergence speed was relatively slow because of not optimized path planning. The path planning for the experiment is divided into two processes, turning robots body and moving straight. However the target keeps moving while player robots are turning at a fixed place to adjust its heading direction. If the target's moving speed is too much faster than players it is almost impossible to catch the target. For this reason speed limit for the target robot was required.

CHAPTER 7

CONCLUSION

7.1 Summary

This thesis presents how potential game based cooperative control can be applied to dynamic environment problems such as 'moving target tracking,' 'sensor coverage in unknown dynamic environments' and 'path optimization in stochastic environments.' The three problems are formulated as potential games and suitable learning algorithms are developed. The developed approach is illustrated through simulation programs.

In the moving target tracking example in Chapter 3, players make predictions about future environmental states. Even though the prediction is not perfect, it is shown that it helps players to adapt to a dynamic characteristic. In the sensor coverage problem in Chapter 4, players estimated utility based on their observations and a notion of fading memory. Lastly, the path optimization scenario in Chapter 5 illustrates payoff based cooperative control in dynamic environments. From the aspect of available information to players, the three examples can be classified as follows:

Table 5: Information availability of examples

	Utility function over joint action set	Other players' realized actions and rewards
Chapter 3	Known	Known
Chapter 4	Unknown	Known
Chapter 5	Unknown	Unknown

Furthermore, a laboratory test bed with supporting hardware, software, and networking was developed to test target tracking. The test bed includes multiple mobile robots, indoor GPS and mobile robot remote control software.

7.2 Future works

The learning algorithms developed in this thesis are variants of algorithms from prior research based on static environments. The performance of the developed learning algorithms was simulated but rigorous mathematical proofs have yet to be derived. This is needed to ensure the learning algorithms' convergence in more general settings.

The converging speed to equilibrium is also an important issue. If an algorithm's converging speed is impractically slow, it cannot be applied to a real application even though its convergence is proved. Therefore further researches about the converging speed are needed to meet practical applications' requirements.

The study about more various problem setting is needed. This research was focused on potential games but more generalized form of games such as weakly acyclic games [22] in dynamic environments can be studied. Or multi agent systems in hostile environments can be considered. There can be a problem in which some adversaries or the environment of the system interfere with optimization of an objective function.

The multi agent test bed of this research also has opportunities for future research. Path planning is an important research field for mobile robots and lots of research have been done. If more advanced path planning is applied then the entire performance can be improved. Furthermore larger number of robots can be considered for future works.

REFERENCES

[1] P. Ogren, E. Fiorelli and N. E. Leonard, "Cooperative control of mobile sensor networks: Adaptive gradient climbing in a distributed environment," *IEEE Transactions on Automatic Control,* 49(8): 1292-1302, 2004.

[2] W. Li and C. G. Cassandras, "Sensor networks and cooperative control," *European journal of control,* 11(4-5): 436-463, 2005.

[3] G. Arslan, J. R. Marden and J. S. Shamma, "Autonomous vehicle-target assignment: a game-thoretical Formulation," *Transactions on ASME, Journal of Dynamic Systems, Measurement, and Control,* 129(5): 584-596, 2007.

[4] R. M. Murray, "Recent Research in Cooperative Control of Multivehicle Systems," *Transactions on ASME, Journal of Dynamic Systems, Measurement, and Control,* 129(5): 571-583, 2007

[5] J. S. Shamma and G. Arslan, "Dimensions of cooperative control," in *Cooperative control of distributed Multi-Agent System,* J. S. Shamma. Ed., NJ: Wiley-Interscience, 2008.

[6] A. Zhu and S. X. Yang, "A survey on intelligent interaction and cooperative control of multi-robot systems," *the 8th IEEE International Conference on Control and Automation,* Pages 1812-1817, 2010.

[7] Y. Shoham and K. Leyton-Brown, *Multiagent systems: algorithmic, game-thoretic, and logical foundations,* Cambridge, U.K.: Cambridge University Press, 2008.

[8] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behaviour*, Princeton University Press, 1944.

[9] S. Parsons and M. Wooldridge, "Game theory and decision theory in multi-agent systems," *Journal of Autonomous Agents and Multi-Agent Systems*, 5: 243-254, 2002.

[10] J. R. Marden, G. Arslan, and J. Shamma, "Cooperative control and potential games," *IEEE Transactions on Systems, Man, and Cybernetics*, 39(6): 1393-1407, 2009.

[11] J. R. Marden, H. P. Young, G. Arslan, and J. S. Shamma, "Payoff based Dynamics for Multi-player weakly acyclic games," *the 46th IEEE Conference on Decision and Control*, Pages 3422-3427, 2007.

[12] D. Monderer and L. Shapley, "Potential games," *Journal of Games and Economic Behavior*, 14(1): 124–143, May 1996.

[13] D. H. Wolpert and K. Tumer, "Collective intelligence, data routing and Braess' paradox," *Journal of Artificial Intelligence Research*, 16: 708-714, 2002.

[14] L. E. Blume, "The statistical Mechanics of strategic interaction," *Journal of Games and Economic Behavior*, 5(3): 387-424, 1993.

[15] V. S. Borkar and P. R. Kumar, "Dynamic Cesaro-Wardrop equilibration in networks," *IEEE transactions on Automatic Control*, 48(3): 382-396, 2003.

[16] "The Khepera III robot," December 5, 2006. [online]. Available: http://www.k-team.com. [Accessed: April 2010].

[17] "Vicon motion capture system," January 12, 2009. [online]. Available: http://www.vicon.com. [Accessed: April 2010].

[18] M. G. Jones, *Design and Implementation of a multi-agent systems laboratory*, Masters thesis, Georgia Institute of Technology, 2009.

[19] J. F. Kurose and K. W. Ross, *Computer Networking*, 5th ed. Boston, MA: Pearson Education Inc., 2010.

[20] F. Aurenhammer, "Voronoi diagrams—a survey of a fundamental geometric data structure," *ACM Computing Surveys (CSUR)*, 1991.

[21] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 4(2): 100–107, 1968.

[22] H. P. Young, *Strategic learning and its limits*, Oxford University Press, 2005.